ICASE

OPTIMAL ASSIGNMENTS IN DUAL-PROCESSOR DISTRIBUTED SYSTEMS UNDER VARYING LOAD CONDITIONS

Shahid Bokhari

Report Number 79-14

July 5, 1979

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia

Operated by the

UNIVERSITIES SPACE USRA RESEARCH ASSOCIATION

OPTIMAL ASSIGNMENTS IN DUAL-PROCESSOR DISTRIBUTED SYSTEMS UNDER VARYING LOAD CONDITIONS

Shahid H. Bokhari
Institute for Computer Applications in Science and Engineering

ABSTRACT

In dual-processor systems the optimal assignment of the modules of a distributed program over the two processors may be found using a network flow algorithm. The optimal assignment is sensitive to the loads on the two processors and is not usually feasible to recompute each time the loads change.

We address the problem of computing all optimal assignments for all possible load values in advance of actual execution of the distributed program. A mathematical model is developed wherein the situation is represented by a convex polyhedron in 3-space, each of whose faces corresponds to a specific optimal assignment. It is proved that, even though there are 2^n distinct assignments of n modules over the two processors and an infinite number of load values, the number of optimal assignments is $O(n^2)$. A fast look-up technique that, given the polyhedron, finds the optimal assignment at a specific pair of loads in $O(n \log n)$ time is described. An algorithm that finds the polyhedron in $O(n^7)$ time is presented.

The results presented here provide a means for adapting rapidly to changes in the load on both machines.

This research was supported by NSF Grant MCS-76-11650, while the author was at the Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, and by NASA Contract No. NAS1-141401 while the author was resident at ICASE, NASA Langley Research Center, Hampton, VA 23665.

I. Introduction

The potential for distributed processing exists wherever there are two or more computer systems that are interconnected in such a fashion that a program (or subprogram) running on one machine can invoke a different program on another machine. Such situations exist in numerous industrial and academic computing environments.

Given the possibility of executing the various modules that make up a large program on different processors, we are naturally interested in doing this in an optimal fashion. Each module should ideally execute on the processor that is best suited to the sort of computation that is being carried out by that module. Modules that communicate heavily with each other should reside, as far as possible, on the same machine. If we associate with each module its execution costs on each of the available processors, and if the costs of communication between pairs of modules (should they not be coresident) are also given, then the problem becomes one of assigning modules to processors such that the sum of execution and communication costs is minimized.

Stone [77a] has shown that this problem may be solved for the two-processor case using a network flow algorithm. An extension to the three processor case is described by Stone [77b]. Rao et al. [79] describe how a memory constraint on one of the processors may be taken into account. Bokhari [79] has extended the network flow model to include the possibility of reassigning modules during the course of program execution, while taking the costs of reassignment into account.

Further research by Stone [78] shows how the assignments may be found efficiently under conditions of varying load on only one processor.

This research addresses the problem of finding the optimal assignments for dual-processor systems under varying load conditions on both processors. In the typical industrial or academic environment, the processors that make up a distributed system are usually time-shared. In such circumstances, the cost of running a module on a specific processor depends on the load on that processor. The optimal assignment of modules to processors is, therefore, sensitive to the load conditions on the processors. The network flow algorithm may, of course, be used to find the optimal assignment at a specific pair of load levels. However, it is usually not feasible to do this in real time because, even though the algorithm is efficient, the time required to solve the assignment problem can equal or exceed the time required to run the program whose execution cost is being minimized.

We present in this paper an algorithm that enables us to compute efficiently all optimal assignments for all possible values of load in advance of actual execution of the distributed modular program. A very fast look-up technique that finds the optimal assignment of all n modules at specified load conditions in O(n log n) time is also developed. The results presented here provide a means for adapting rapidly to changes in the load on the two machines.

This paper is organized as follows. In Section II we show that, assuming the run costs of modules to vary linearly with the load on the

processors, the cost of optimal assignments at all values of load correspond to a convex polyhedron in 3-space. In Section III we show how all the relevant information about the polyhedron is contained in its projection on the XY plane. The total number of faces on the polyhedron is proved to be $O(n^2)$ in Section IV. In Section V we develop the concept of critical load lines--these permit us to look up the optimal assignment at any value of load in O(n log n) time. An algorithm for finding the polyhedron is presented in Section VI. We conclude with a discussion on implementation considerations and describe some new problems generated by this research.

II. The Assignment Polyhedron.

In this section we present our assumptions and develop a mathematical model for our assignment problem. We first briefly review the network flow algorithm for the solution of the assignment problem for given load conditions and the extension to the case where the load on only one processor is variable. We go on to show that, under assumptions of linearity, the cost of each assignment under varying load conditions on both processors corresponds to a plane in 3-space and that the optimal assignments form a convex polyhedron, such that each face of the polyhedron corresponds to a specific optimal assignment.

The following is a brief exposition of Stone's network flow model [Stone,75]. A distributed program is considered to be a collection of modules (which may be subroutines, coroutines, or data files). A module may, in general, execute or reside on either of the two processors. For each module we are given the cost of executing it on either processor. For each pair of modules we are given the cost of interprocessor communication between them, should they not be coresident. Costs may be measured in time, dollars or other resource units.

The problem of finding the assignment that minimizes the net cost is solved by drawing up an assignment graph in which each cut corresponds to an assignment and the weight of the cut equals the cost of the assignment. By applying a maxflow algorithm to this graph, we

may obtain the minimum weight cut and hence the minimum cost assignment. This algorithm gives us the optimal assignment for fixed values of loads on the two machines. Should the loads change, the algorithm may be run again with suitably modified values of the runcosts. This is viable if the loads change after periods of time that are much longer than the time required to run the algorithm. Since this is often not the case, we would like to have procedures that enable us to compute all optimal assigments prior to the execution of the distributed program and to have means for rapidly looking up the optimal assignment at a given load level.

This problem has been solved by Stone [78] for the case where the load on only one machine varies. It has been shown that for increasing values of load on one machine, successive optimal assignments are nested and hence the total number of assignments for a problem involving n modules is no more than n+1. The results presented below solve the problem for variable loads on both machines and may be considered to be a two-dimensional extension of the previous research.

Let us label our two processors X and Y. We assume that the load levels on the two machines may be described by the real positive numbers x and y respectively. A specific pair of load levels <x,y> is called a load point and may lie anywhere in the positive XY plane.

The cost of running a module A on processor X is assumed to vary linearly with the load on X. The constant of proportionality is denoted a_x. Thus the cost of running module A on processor X is

$a_x x$ and, similarly, the cost for running A on Y is $a_y y$.

Note. For clarity of presentation, we have assumed that x and y vary from 0 to infinity, thereby implying that at load zero each module executes in zero time. This should create no difficulties in the implementation of this work because loads will in practice be constrained to lie between some x_{min} and x_{max} and some y_{min} and y_{max}. Thus the shortest time in which a module can execute on processor x would be $a_x x_{min} > 0$.

The cost of communication between a pair of nodes, should they not be coresident, is assumed to be constant over all possible values of load. This assumption is justified by practical experience with the Brown University Graphics system [Michel & van Dam 77] where the network flow algorithm was first implemented.

Consider the cost figures shown in Table I. Here we have a problem involving 5 modules. The run costs of each module on each processor are given as variables in x and y. The communication costs are for the indicated pair of modules, should they not be coresident. These costs may be inserted into the assignment graph of Fig. 1, as described by Stone [77a]. Each cut in this graph corresponds to an assignment and vice-versa. The weight of each cut equals the cost of the corresponding assignment. For example, the thick line in Fig. 1 shows a cut that assigns modules 1 & 2 to processor X and the remaining modules to processor Y. The weight of this cut is Z=12x+16y+16, this being the sum of the weights on the constituent edges. In general, a

	Runcost On	Runcost On
Module	X	Y
1	10x	11y
2	2x	37y
3	x	5y
4	∞	7y
5	4x	4y
	x=load on X	y=load on Y

Communication Costs

Module	1	2	3	4	5
1	-	3	7	8	0
2		-	0	1	0
3			-	0	7
4				-	9
5					-

TABLE I. Runcosts and Communication Costs for a 5 module problem.

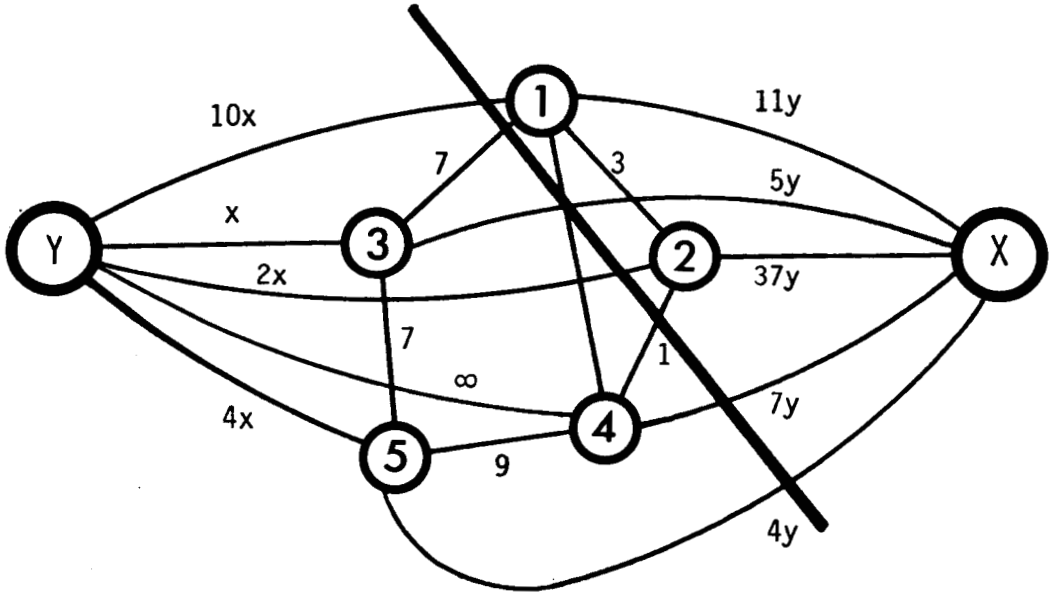

Fig. 1 Assignment Graph for the Problem shown in Table I

cut will have weight z=Mx+Ny+C, where M(N) equals the sum of run costs of modules assigned to X(Y). C equals the sum of communication costs between modules that are not coresident.

Since there is a one-to-one correspondence between cuts and assignments, we may identify a cut by listing the set of nodes it assigns to processor X (the remaining nodes must necessarily be assigned to processor Y). Thus the cut indicated in Fig. 1 is denoted {1,2}.

We will call the equation of the weight of a cut (or assignment) just the equation of a cut (or assignment) for brevity. Thus the equation of cut {1,2} is z=12x+16y+16 and the equation of cut {1,2,3,5} (not marked in the figure) is z=17x+7y+18. The cardinality of a cut or assignment is the cardinality of the set of nodes it assigns to processor X.

The assignment that puts no modules on X is called the Null assignment, denoted $\emptyset$, and has an equation of the form z=Ny. The assignment that puts all modules on X is called the Universal assignment, denoted U, and has an equation of the form z=Mx.

The general equation of a cut z=Mx+Ny+C represents a plane in 3-space. Since there are 2^n possible assignments for a problem involving n modules and two processors, we will have 2^n such equations of planes. The constants M,N and C are all non-negative (because execution and communication costs are all non-negative) and,

as a result, none of these planes can intersect with the XY plane within the positive quadrant. Every plane has a non-negative intercept with the Z-axis (since $C \geq 0$). The slopes of these planes in the X and Y directions are also nonnegative (since $M,N \geq 0$).

We will henceforth call the positive quadrant of the XY plane the load plane because each point on it represents a load point. At a given load point <x,y>, the lowermost of the 2^n planes (the plane with the smallest z-coordinate) is the one that represents the optimal assignment. This may be found by substituting the specific values of x and y into the assignment graph and applying a network flow algorithm to it. As we are interested only in optimal assignments, we need consider only the lowermost plane at each <x,y>. A little reflection will reveal that these lowermost planes intersect to form a convex polyhedron whose faces correspond to specific assignments. We call this the assignment polyhedron because each face on this polyhedron represents an assignment that is optimal for all load points <x,y> that lie within its projection on the XY plane. Fig. 2 shows a typical assignment polyhedron.

It is clearly not feasible to apply a network flow algorithm at each point on the load plane in order to find the polyhedron. Subsequent sections of this paper discuss how the polyhedron may be found efficiently and also how it may be used to solve our assignment problem.

III. The Load Plane.

The assignment polyhedron described in the previous section is convex. It follows that each individual face is a convex polygon and further that the projection of the polyhedron onto the XY plane (the load plane) is made up of convex polygons.

The load plane is thus dissected into convex polygonal regions each of which, having been projected by a face, corresponds to an assignment that is optimal for all load points that fall within it, and is labelled accordingly. Fig. 3 shows the load plane corresponding to the polyhedron of Fig. 2, which is derived from a graph having eight module nodes.

Note. The regions at the extreme right and top of the load plane are unbounded (see property 3.7 below).

The "nesting" theorem by Stone [78] states that in a distributed processing situation of the sort being considered here, if the load on only machine Y increases the optimal assignment will change such that modules move away from machine Y onto machine X and only in that direction. In other words, successive optimal assignments are nested. This is clear from Fig. 3. The vertical dashed line, parallel to the Y axis, represents an increase in the load on processor Y, while the load on processor X is maintained at a constant value. Traveling in the positive Y direction, we encounter regions $\emptyset$, {8}, {7,8}, {4,7,8}, {3,4,7,8}, {3,4,6,7,8}. The corresponding property holds if we keep

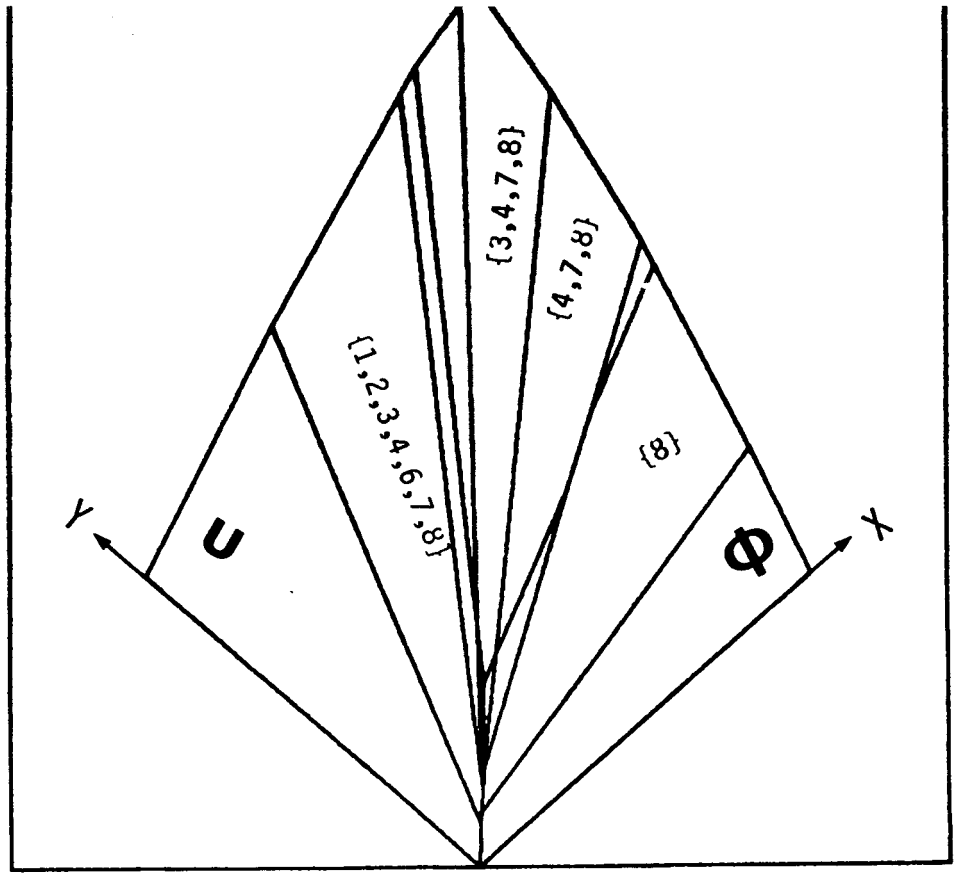

Fig. 2 3-Dimensional view of an Assignment Polyhedron for a problem involving 8 modules. For clarity, some of the faces have not been labelled.

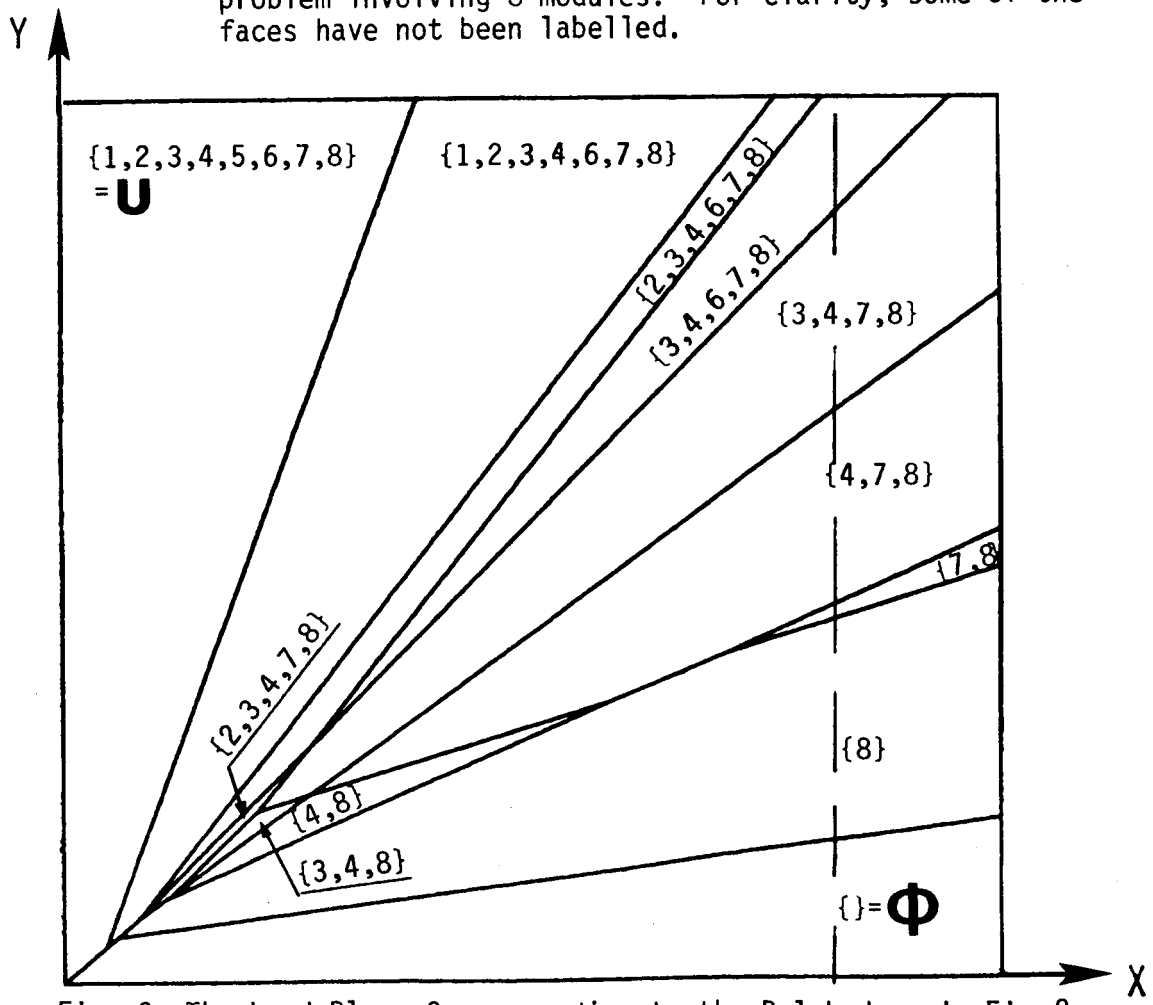

Fig. 3 The Load Plane Corresponding to the Polyhedron in Fig. 2

the load on Y constant and vary the load on X.

The load plane has the following properties.

3.1 A horizontal or vertical line through the load plane cannot cut through more than n+1 regions (n is the number of modules). This follows from the nesting theorem.

3.2 Each point $\langle x,y \rangle$ on the load plane falls within a region that represents an assignment that is equal to or contained in the assignment at $\langle x-\Delta x, y+\Delta y \rangle$ $(\Delta x, \Delta y > 0)$ (Fig. 4). This also follows from the nesting theorem.

3.3 The lines defining the regions of the load plane all have positive slope. This follows from property 3.2. A line with negative slope cannot separate two regions. For example, in Fig.5, suppose $\langle x_1,y_1 \rangle, \langle x_3,y_3 \rangle$ are contained in one region and $\langle x_2,y_2 \rangle$ in another, and a line of negative slope separates the two regions. Then the assignment at $\langle x_1,y_1 \rangle$ must be contained in the assignment at $\langle x_2,y_2 \rangle$, which in turn must be contained in $\langle x_3,y_3 \rangle$. This is possible only if all three points belong to the same region. (This property may also be established using arguments based on the convexity of the polyhedron and the properties of the equations of the planes.)

3.4 Any continuous or piecewise continuous curve that has negative

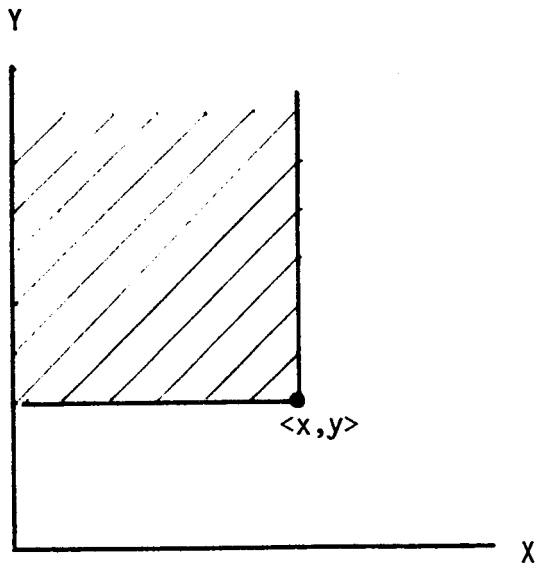

Fig. 4. Property 3.2: The assignment at <x,y> must be equal to or contained in every assignment in the shaded region

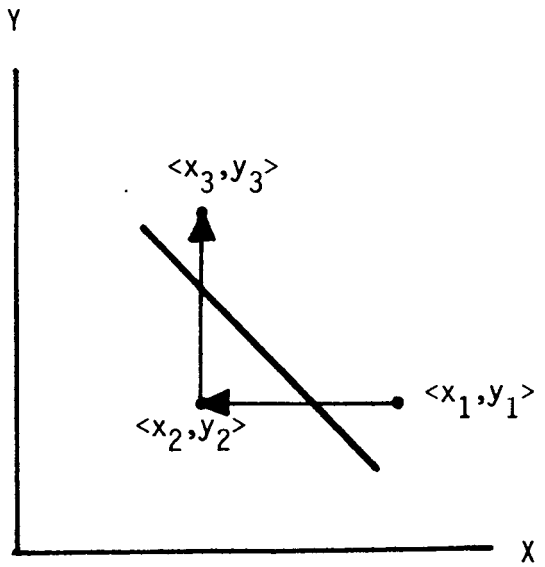

Fig. 5 Property 3.3: The line separating two regions cannot have negative slope.

slope everywhere along its length must pass through successively nested regions (Fig. 6). Clearly, this curve cannot pass through more than n+1 regions.

3.5 If we represent each region in the load plane by a node and draw a directed edge between nodes representing adjacent regions in the direction of nesting, we obtain a planar lattice, as shown in Fig. 7. Obviously, no path in this lattice has length greater than n+1.

3.6 The X-axis can be touched only by the $\emptyset$ region for all $x>0$. This is because the equation for the $\emptyset$ region is $z=Ny$ which is zero for all $y=0$ and all other regions have $z>0$ for $y>0$. Similarly, the Y-axis can be touched only by the U region, for all $y>0$.

3.7 The regions for very large values of x and y are all unbounded. These "fringe" regions include the $\emptyset$ and U regions.

We may at this point be tempted to conclude that properties 3.1 and 3.2 are sufficient to prove that the total number of regions is no more than n^2. While the number of regions will be proved to be $O(n^2)$ in the next section, this does not necessarily follow from the properties listed above. It is possible to construct "load planes" having 2^n regions that have all the above properties.

We conclude this section by observing that, because there is a

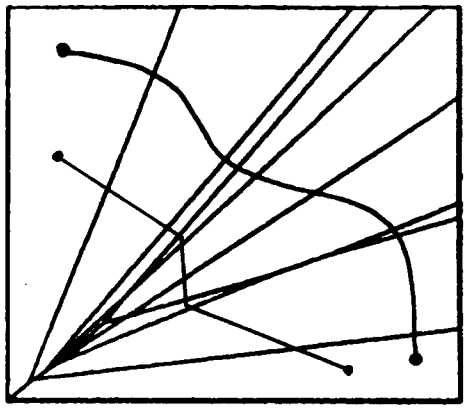

Fig. 6 Property 3.4: Any curve that has negative slope throughout its length cannot pass through more than n+1 regions.

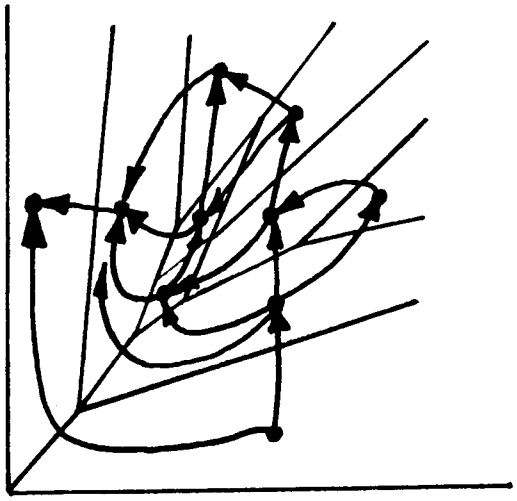

Fig. 7 Property 3.5: The nesting relationships between regions form a planar lattice.

one-to-one correspondence between regions on the load plane and the faces on the polyhedron, the properties listed above apply in an obvious fashion to the polyhedron itself.

IV. The Maximum Number of Distinct Assignments.

In this section we show that the maximum number of regions on the load plane (and hence the number of faces on the polyhedron) is $O(n^2)$. We introduce the concept of elemental polyhedra, these being polyhedra whose intersection is the assignment polyhedron and whose properties permit us to obtain the bound on the number of regions on the load plane.

Let us consider an assignment graph of the type shown in Fig. 1, and concentrate our attention on two cuts A and B such that $A \not\subseteq B$ and $B \not\subseteq A$. We can condense our graph into four "Supernodes" X′, Y′, A′ and B′, where X′(Y′) represents the processor node X(Y) plus all nodes assigned to X(Y) by both A and B (Fig. 8). Supernode A′ represents all nodes assigned to X by A and to Y by B. Similarly, B′ represents all nodes assigned to X by B and to Y by A. The edges between the supernodes have weights equal to the sum of the weights on the edges between the constituent nodes.

There are four possible cuts in this graph: the cuts A and B described above; the cut $\emptyset$ which represents the assignment of nodes $A \cap B$ to X; and the cut U which represents the assignment of nodes $A \cup B$ to X.

The polyhedron generated by these four cuts is called an elemental polyhedron. It should be clear that each pair of cuts A,B ($A \not\subseteq B, B \not\subseteq A$) in an assignment graph will give rise to an elemental polyhedron. The

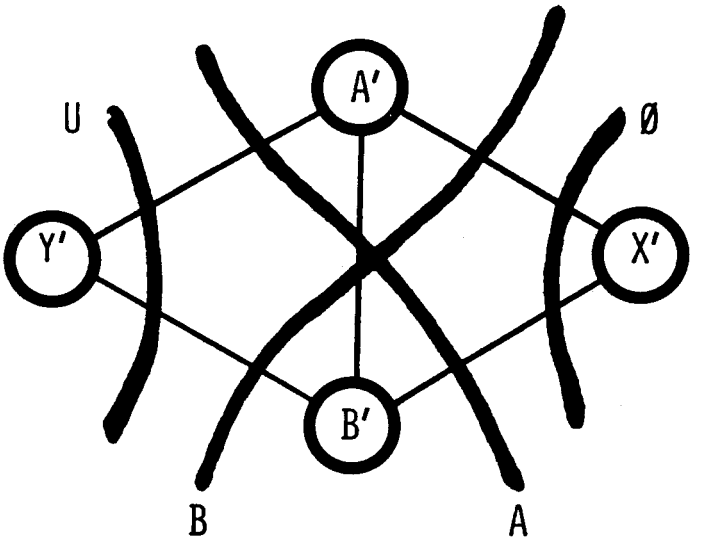

Fig. 8 A condensed Graph

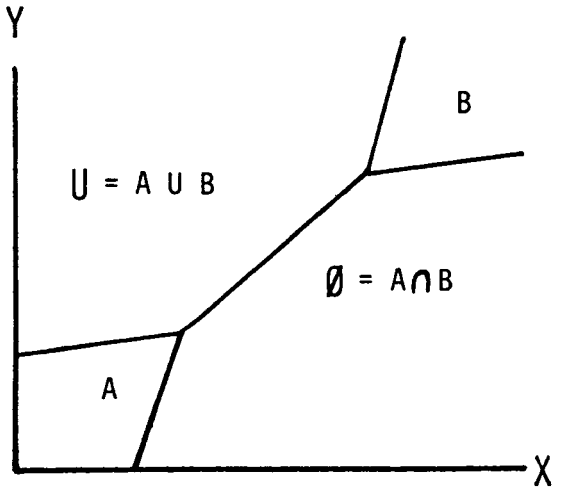

Fig. 9 The elemental load plane

assignment polyhedron corresponding to a particular graph is made up of the intersection of all the elemental polyhedra generated by all pairs of cuts. The load plane corresponding to an elemental polyhedron is called an elemental load plane (Fig. 9).

Note. The elemental load plane shown in Fig. 9 has the maximum number of regions on it (that is, four). This is not always the case and some elemental load planes may have either A or B or both missing. In the condensed graph, the equation of the assignment $U(\emptyset)$ does not have the form $z=Mx$ ($z=Ny$). This is because each edge is a condensation of several edges from the original graph and thus its weight is the sum of several communication and execution costs. For this reason the elemental load plane does not obey property 3.6--the X and Y axes may be touched by two regions each.

The faces corresponding to A and B can have at most one point in common (that is, they can at most touch at a point). They can never share a common edge because $A \not\subseteq B$ and $B \not\subseteq A$ by definition. Thus we see that the region $U = A \cup B$ will always be to the left of and above the regions A and B. Similarly the region $\emptyset = A \cap B$ will always be to the right of and below A and B. This may formally be stated as follows.

4.1 In the elemental load plane every point in the region $A \cup B$ may be connected by a straight line to some point in region A(B) such that the straight line does not pass through region B(A). The same applies to region $A \cap B$.

This excessively formal way of stating an obvious fact is, unfortunately, essential to the proof that follows. As the assignment polyhedron is made up of the intersection of elemental polyhedra, it follows that the abovementioned property of the elemental load plane must carry over to the load plane corresponding to the complete assignment polyhedron. This permits us to prove the following theorem:

Theorem 4.2. The number of regions (faces) on the load plane (assignment polyhedron) is $O(n^2)$.

Proof. The maximum number of regions of cardinality 1 is n. For the case n=6 this is shown in Fig. 10, where the number of faces of cardinality 2 is n-1=5. Any further regions of cardinality 2 must lie in the shaded regions so as not to violate property 3.2, but this would cause property 4.1 to be violated. It follows that there cannot be greater than n-1 regions of cardinality 2. Similarly the number of regions of cardinality 3 cannot exceed n-3, and so on. Thus the total number of regions is $n(n+1)/2 + 1$. This proves the theorem.

Corollary 4.3. The number of straight line segments (edges) on the load plane (assignment polyhedron) is $O(n^2)$.

Proof. Follows from Theorem 4.2 and a result by Euler[1752]: On a polyhedron in 3-space the number of edges plus six cannot exceed three times the number of faces.

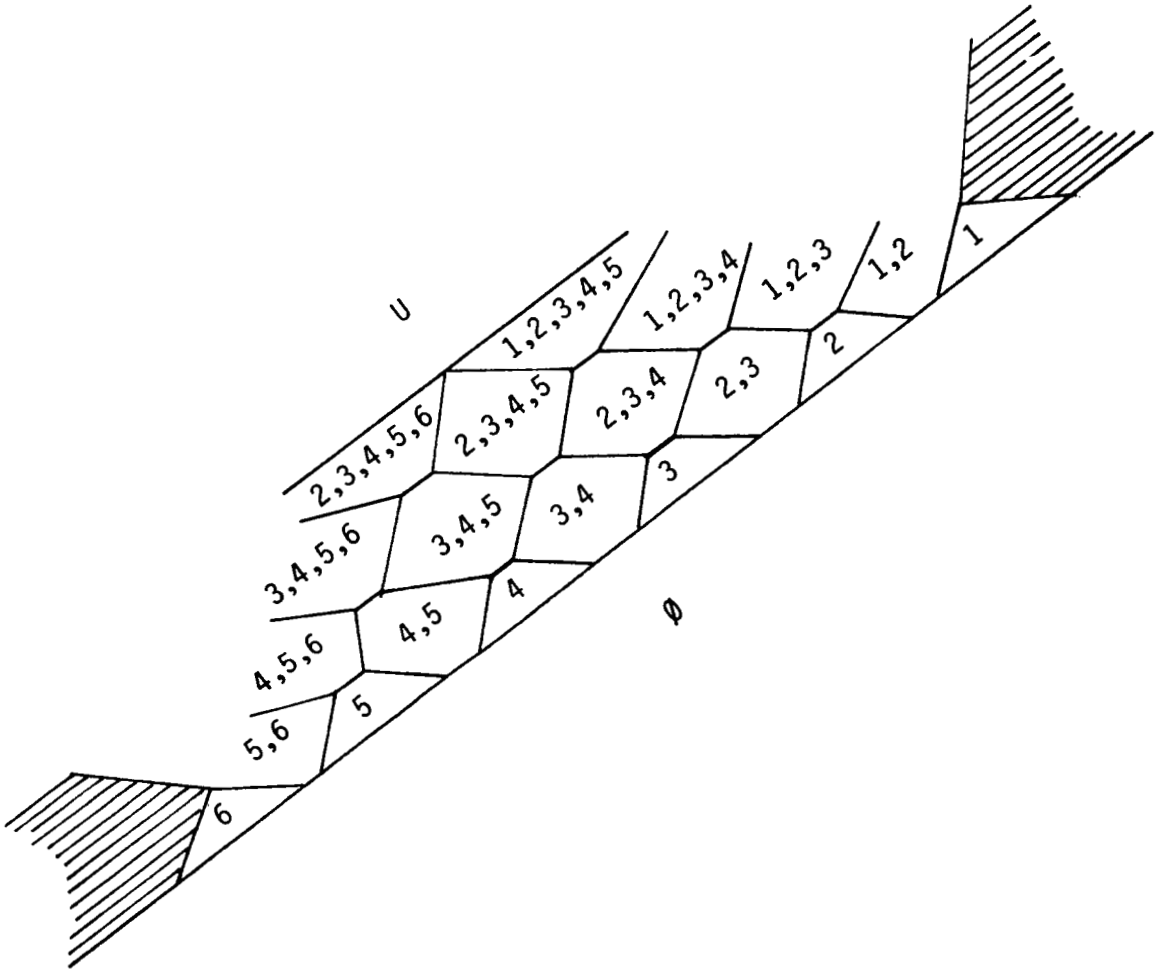

Fig. 10 The maximum number of regions

There are several other properties of the assignment polyhedron that can be established and shown to reduce the total number of regions further. However these properties do not reduce the polynomial bound on the number of regions below n^2 and are therefore ignored here.

V. Critical Load Lines.

Given the load plane described above, we obviously need some means for utilizing it in our assignment problem. In this section we present an efficient technique for finding the assignment of all modules of the program, given the load point.

Let us examine the load plane shown in Fig. 11, concentrating our attention om module 3. We observe that the load plane may be divided into two super regions such that if the load point is in one region, module 3 is assigned to processor X and if in the other, it is assigned to processor Y. In Fig. 11, the super region comprising regions {3}, {2,3}, {3,4}, {2,3,4}, {3,4,5}, etc. is the one for which module 3 is assigned to processor X. Clearly, one such division of the load plane exists for each module. This leads to the following concept.

Definition. The critical load line for a module is a piecewise continuous line on the load plane such that if the load point falls to the right of (or below) this line, the optimal assignment will assign that module to processor Y. If the load point falls to the left of (or above) this line, the module is assigned to processor X by the optimal assignment.

The thick line shown in Fig. 11 is the critical load line for module 3.

The following properties of the critical load line may be

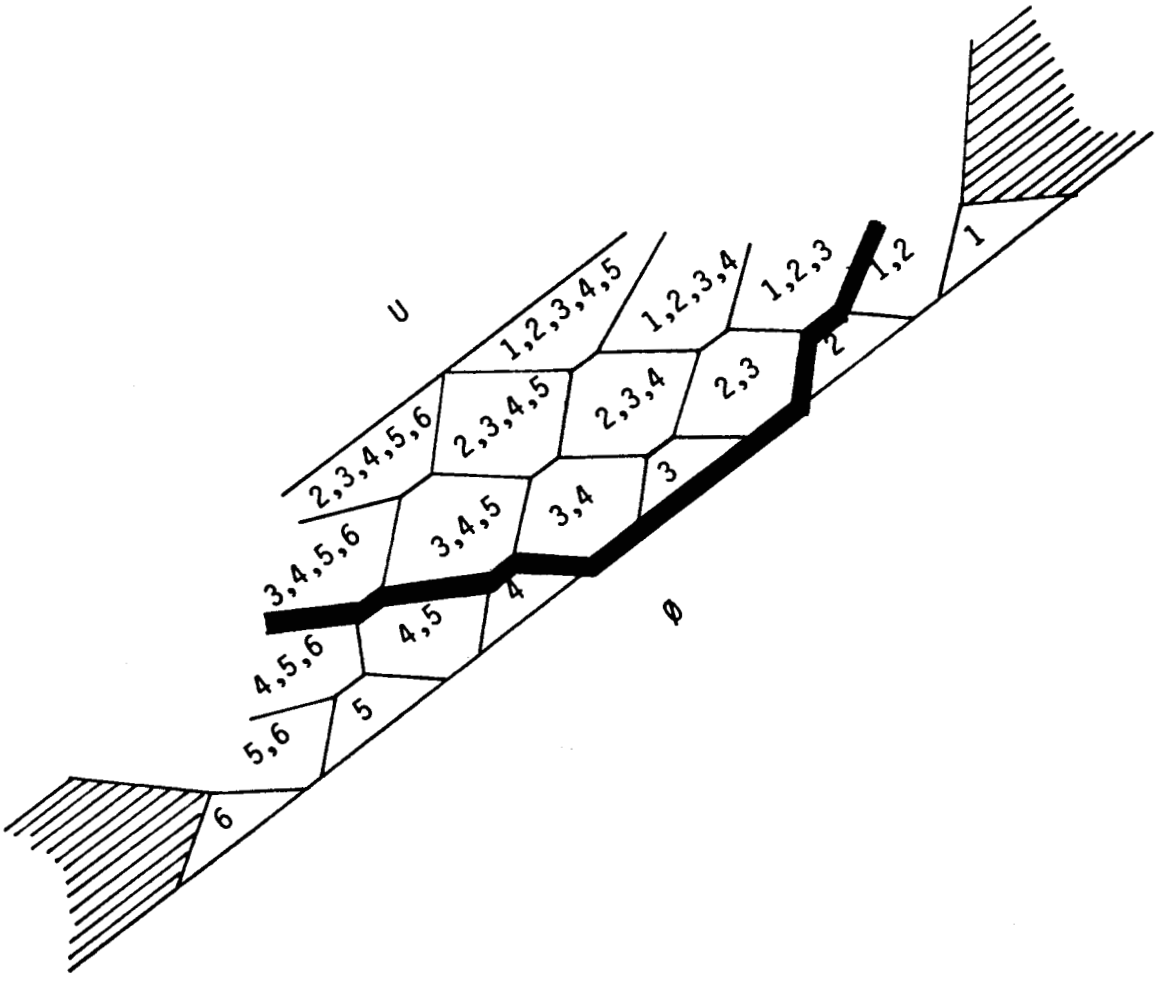

Fig. 11 Critical load line for module 3.

enumerated.

5.1 A critical load line starts at the origin and continues indefinitely.

5.2 Each segment of a critical load line is a straight line with positive slope. (Negative slopes would violate the nesting property).

5.3 The y-coordinate of the critical load line increases monotonically with the x-coordinate (follows from 5.2).

5.4 Every critical load line has $O(n^2)$ segments. This follows from corollary 4.3: the number of segments in a critical load line is surely less than the total number of segments on the entire load plane.

5.5 At least two critical load lines pass through the point of intersection of 3 regions on the load plane (Fig. 12). This point of intersection is, in fact, a vertex of the assignment polyhedron.

5.6 The set of n critical load lines, one for each module, completely specifies the load plane. If the z coordinate is included at each breakpoint of each critical load line, then the set of critical load lines completely specifies the assignment polyhedron.

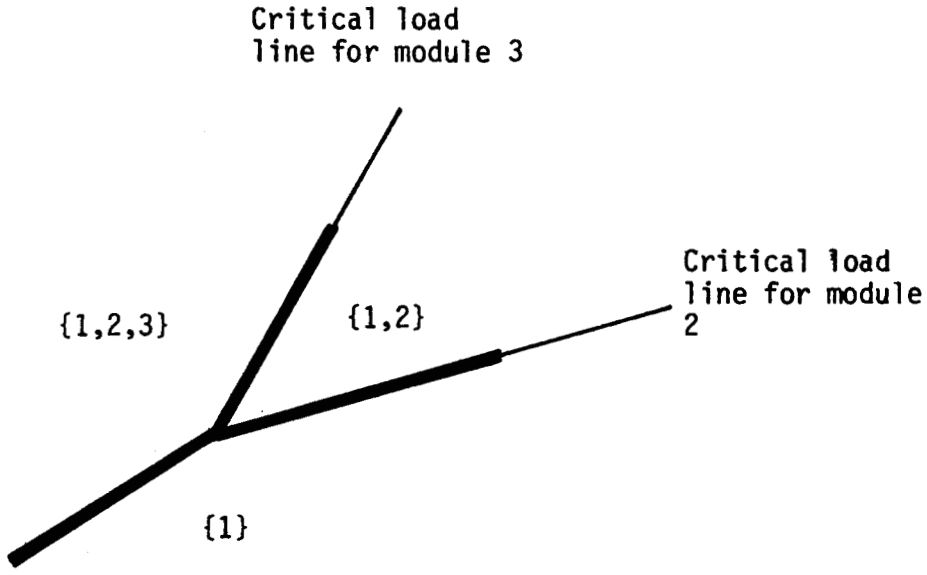

Fig 12. At least two critical load lines pass through a point of intersection of three regions.

The critical load line associated with a particular module may be used to find the optimal assignment for that module for a given load point. Suppose we are given a critical load line (Fig. 13) specified by the end points of its segments. Then we may determine whether a load point $\langle x_1, y_1 \rangle$ lies above or below the line as follows.

1. Divide the load plane into vertical strips as determined by the end points of the segments of the critical load line.

2. Determine which strip the x coordinate of the point $\langle x_1, y_1 \rangle$ lies in. Since the y coordinate of a critical load line increases monotonically with x (property 5.3), we may use a binary search at this step.

3. Determine whether the y coordinate lies above or below the segment within the strip found in step 2.

The number of segments being $O(n^2)$, the search in step 2 can be done in O(log n) time. Step 3 takes constant time. The determination of the optimal assignment for all n modules will thus take O(n log n) time.

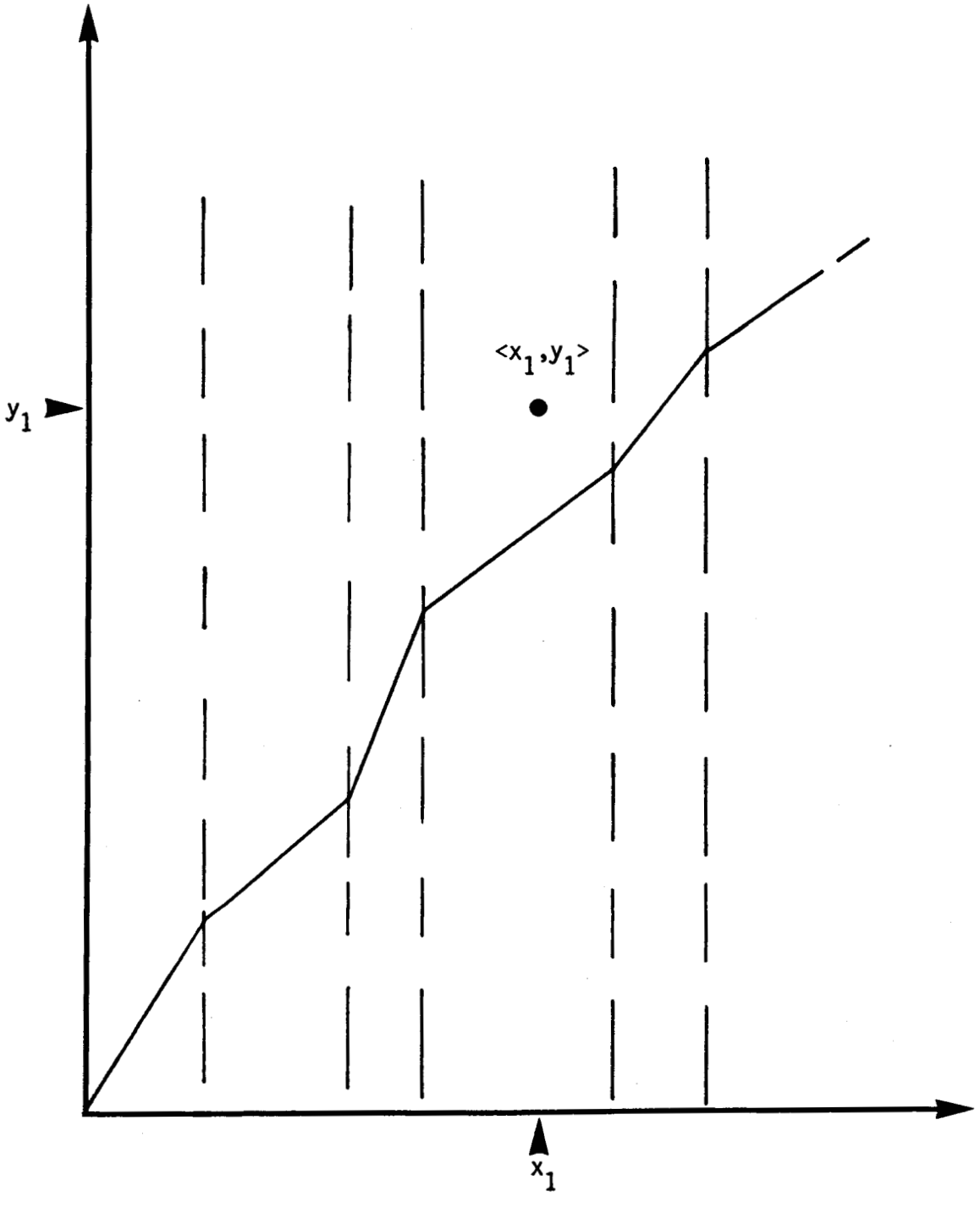

Fig. 13 Determining whether a point $<x_1,y_1>$ lies above or below a given critical load line.

VI. An Algorithm for finding the Assignment Polyhedron.

The algorithm described in this section will accept the assignment graph and the minimum and maximum values of loads x and y and will find all critical load lines in $O(n^4)$ appplications of the maxflow algorithm.

We assume, for purposes of exposition, that the following data structures, functions and procedures are available to us.

Function MAXFLOW(x,y:real):nodeset; Will substitute the values x and y in the given assignment graph, apply a maxflow algorithm to it, obtain the mincut and return the set of nodes assigned by this mincut to processor X. Recall that this set of nodes uniquely specifies an assignment and that the equation of the plane corresponding to this cutset can be obtained from the assignment graph.

Data Structure PAIRLIST; will maintain a list of pairs of adjacent faces on the polyhedron that have been examined so far by the algorithm.

Procedure INSERT_PAIR (A,B:nodeset); will insert the pair of assignments A,B into PAIRLIST.

Function IN_PAIRLIST(A,B:nodeset):boolean; returns true if A and B are already on PAIRLIST.

Function LOWEST_PLANE(A,B:nodeset):nodeset; This function will find the line of intersection of planes A and B and establish the highest point on this line that is on the hull. It will return the identity of the plane that intersects with A and B at this point (that being the lowest plane intersecting this line). It first finds the furthest point on the line of intersection of A and B that is within the region defined by x_{max} and y_{max}. It then finds the mincut C at this point. The point of intersection of A,B and C is then considered the new furthest point and the process repeated until C=A or C=B, whereupon C is returned as the lowest plane. Fig. 14 shows a view of this process in terms of its projection on the load plane.

Data Structure CRITICAL_LINES. The n critical lines will be stored in this structure. During the progress of the algorithm points that lie on specific critical load lines will be inserted into the structure using the following procedure.

Procedure INSERT_CRITICAL(A,B,C:nodeset); The input to this procedure are the 3 planes A,B,C which intersect to form a vertex of the hull. The procedure finds the x,y and z coordinates of the point of intersection and inserts this information into the data structure CRITICAL_LINES.

In addition, we have a stack that holds nodesets and the usual functions, push, pop and empty.

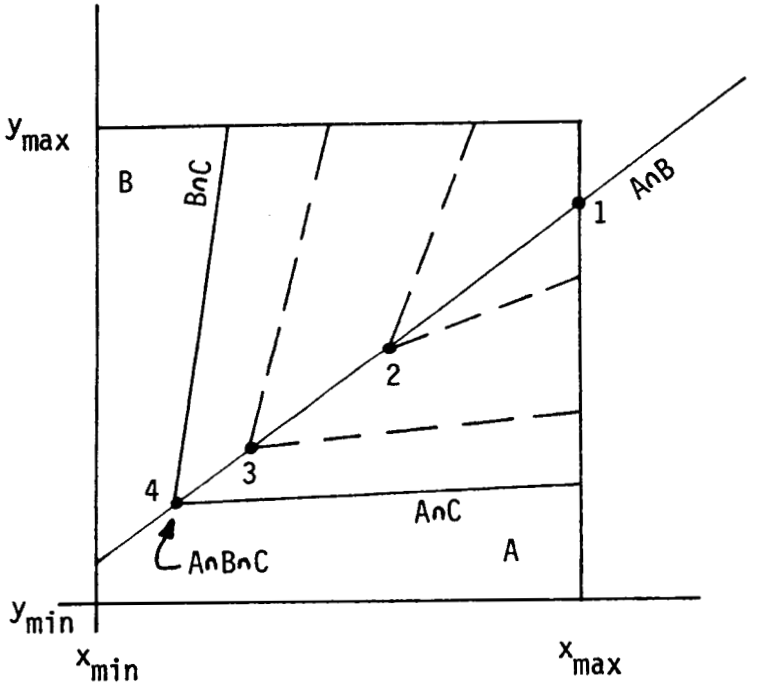

Fig 14. Determining the lowest plane, C, that intersects with the line of intersection of A and B. In this example the algorithm applies the maxflow routine four times before locating the point of intersection of A, B and C.

The Algorithm

```
begin
   Input the assignment graph, x_min,x_max,y_min,y_max;
   A:=mincut(x_max,y_min);
   B:=mincut(x_min,y_max);
   push(A);
   push(B);
   while not empty do
      begin
         pop(A);
         pop(B);
         INSERT_PAIR(A,B);
         C:=LOWEST_PLANE(A,B);
         INSERT_CRIT(A,B,C);
         if not IN_PAIRLIST(A,C) then
            begin
               push(A);
               push(C)
            end;
         if not IN_PAIRLIST(B,C) then
            begin
               push(B);
               push(C)
            end;
       end;
   output critical lines;
end.
```

The algorithm looks at each pair of adjacent faces on the polyhedron once. It executes the function LOWEST_PLANE once for every such pair. The number of adjacent pairs of faces on the polyhedron equals the total number of edges, which is $O(n^2)$. Each call to LOWEST_PLANE will result in a number of executions of the maxflow algorithm. This cannot be greater than the total number of regions on the plane, which is $O(n^2)$. Thus the total number of applications of the maxflow algorithm is $O(n^4)$. The complexity of the best maxflow algorithm being $O(n^3)$ [Karzanov 74], the overall time complexity of the polyhedron finding algorithm is $O(n^7)$.

VII. Discussion and Conclusions.

We have presented a mathematical model for the problem of computing all optimal assignments for the dual processor assignment problem under varying load conditions on both processors. We have shown that the situation can be modelled by a convex polyhedron in 3-space in which each face corresponds to a specific optimal assignment. The total number of faces and hence the total number of distinct optimal assignments was proved to be $O(n^2)$. A fast lookup technique for finding the optimal assignment in $O(n \log n)$ time was developed. The last section presented an algorithm to find the polyhedron in $O(n^7)$ time.

The viability of using network flow algorithms for finding optimal assignments in a practical environment has been established by the Brown University Graphics System [Michel & van Dam 76], [van Dam et al. 74]. Stone's basic assignment algorithm and the nesting property have both been used to advantage in that system. The algorithm presented in this paper extends previous results to the case where varying loads on both machines can be taken into account and is thus applicable to more general situations.

The algorithm for finding the polyhedron, which was described in the previous section, has been implemented and tested on about 100 random graphs with 10 to 20 nodes each. The polyhedron shown in Fig. 2 and the load plane of Fig. 3 were both plotted directly by this algorithm.

Among the open problems generated by this research are the following.

1. How may this approach be extended to three processors? An algorithm for the basic 3-processor assignment problem has been developed by Stone [77b]. Can the nesting properties described in this paper be extended in some way to the three processor case?

2. We have assumed that as the load changes, relocations may be carried out without penalty. What if relocations take an appreciable amount of time? Bokhari [79] describes an extension of the network flow algorithm that takes reassignment costs into account. Can this be extended to the variable load case?

VIII. Acknowledgements

During the course of this research, it has been my pleasure to interact fruitfully with numerous colleagues and mentors. I am especially grateful to Professor Harold Stone for his encouragement of this research and for numerous stimulating discussions on various aspects of the problem. I am also grateful to Azmi Jafarey for developing counterexamples to some early conjectures about the

polyhedron. Discussions with Drs. Milton E. Rose, Robert G. Voigt, J. Gopal Danaraj and Tom Anderson have been very helpful to me in this research.

IX. References

Bokhari [79] S. H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," IEEE Trans. Software Eng., July 1979, to appear.

Euler [1752] Leonhard Euler, "Elementa Doctrinae Solidorum," Novi commentarii academiae scientiarium Petropolitanae, 4 (1752/3), pp. 109-140. Available in Leonhardi Euleri Opera Omina, Series 1, vol. 26: Commentationes Geometricae, pp. 71-93, Lausanne, 1953.

Karzanov [74] A. V. Karzanov,"Determining the Maximal Flow in a Network by the Method of Preflows," Soviet Math. Doklady, vol. 15, no. 2, pp. 434-437, 1974.

Michel & van Dam [76] J. Michel and A. van Dam, "Experience with Distributed Processing on a Host/Satellite Graphics System," Proceeings of SIGGRAPH '76, available as Computer Graphics (SIGGRAPH newsletter), vol. 10, no. 2, 1976.

Michel & van Dam [77] J. Michel & A. van Dam, personal communication.

Rao et al. [79] G. S. Rao, H. S. Stone and T. C. Hu,"Assignment of Tasks in a Distributed Processor System with Limited Memory," IEEE Trans. Computers, vol. C-28, no. 4, pp. 291-299, April 1979.

Stone [77a] H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Trans. Software Eng., vol. SE-3, no. 1, pp. 85-93, Jan. 1977.

Stone [77b] H. S. Stone, "Program Assignment in Three-Processor Systems and Tricutset Partitioning of Graphs," Tech. Rep. no. ECE-CS-77-7, Dept. of Elec & Computer Eng., Univ. of Massachusetts, Amherst.

Stone [78] H. S. Stone, "Critical Load Factors in Distributed Systems," IEEE Trans. Software Eng., vol SE-4, no. 3, pp. 254-258, May 1978.

van Dam et al. [74] A. van Dam, G. Stabler, and R. Harrington, "Intelligent Satellites for Interactive Graphics," Proc. of the IEEE, vol. 62, no. 4, pp. 83-92, April 1974.